



K2 blackpearl Best Practices

ADVICE AND INSTRUCTIONS FOR CHOOSING, DESIGNING, DEPLOYING AND MAINTAINING PROCESS-DRIVEN APPLICATIONS WITH K2 BLACKPEARL

Originally Published November 14th, 2008
Updated October 26th, 2009





INTRODUCTION

In this whitepaper you will find practical guidance on best practices based on methods used by experienced blackpearl designers. This content will be updated as additional best practices are identified.

CONTENTS

- THE PARADIGM SHIFT.....5
 - > Declarative Templates5
 - > Actions and Outcomes5
 - > SmartObjects5
- K2 BLACKPEARL INSTALLATION, CONFIGURATION AND SECURITY BEST PRACTICES6
- SMARTOBJECT DESIGN BEST PRACTICES8
 - > Business Object Centric11
 - > Pros11
 - > Cons11
 - > Process Centric (many process data fields)11
 - > Pros11
 - > Cons11
 - > InfoPath Centric (everything is in the InfoPath XML field)11
 - > Pros11
 - > Cons11
- PROCESS SELECTION12
 - > Process Questions12
 - > People Questions12
 - > Infrastructure Questions12
- PROCESS DESIGN BEST PRACTICES13
- PROJECT STRUCTURE14
 - > Naming K2 Items15
 - > Graphical Design Recommendations15
 - > Source Control16
 - > Process Data17
 - > Process Data Fields17



- > Activity Data Fields17
- > XML Data Fields and InfoPath Forms18
- > Data on Demand18
- > Data Auditing.....19
- > Data Logging.....20
- > Rules20
 - > Preceding vs. Start Rules (Make sure you know the order and what they're used for)20
 - > Line Rules (try to keep maintenance low)20
 - > Escalations (Nothing In Excess).....21
 - > Destination Rules (Know Thyself)21
 - > Succeeding Rules and Activity Scoping22
- > Using GoTos22
- > Process Versioning23
- > Process Sizing.....23
 - > Using Inter Process Communication Events23
- PROCESS PROGRAMMING AND DEBUGGING26
 - > Handling Connections26
 - > Using Console.WriteLine and Logging31
 - > Exception Handling31
- PROJECT DEPLOYMENT36
- CONCLUSION37
- GLOSSARY37

THE PARADIGM SHIFT

If you have previous experience designing processes with K2.net 2003, be prepared to shift your thinking when approaching process design with K2 blackpearl. There are a few key differences between the two versions that are necessary to keep in mind when thinking about process design.

DECLARATIVE TEMPLATES

K2 blackpearl templates differ in a key way from K2.net 2003 templates, namely that they are declarative in nature. This means that the blackpearl templates do not generate code as the K2.net 2003 templates do. Based on Workflow Foundation (WF) schedules, the blackpearl templates use the WF schedules to perform the actions specified in the wizard in a declarative rather than a compiled manner. Values specified in the wizards are substituted for the placeholder values in the templates at runtime. The K2 Server uses the same templates that are available in the K2 Designers to perform a just-in-time (JIT) compilation of the deployed process at runtime. When you modify the code behind a WF schedule at design time, this modified code is persisted in the project file (.kprx) and used to JIT the process at runtime. This mechanism offers a more dynamic runtime of blackpearl processes compared to the K2.net 2003 fully-compiled mechanism, and will affect the way you design processes.

ACTIONS AND OUTCOMES

In K2 blackpearl, actions and outcomes allow process designers to separate what decisions, or Actions, users are able to perform, from the results, or Outcomes, of those decisions. By default there is a one-to-one mapping of Actions to Outcomes, but this can be modified using line rules. Separating Actions from Outcomes allows processes to be modified without the need to modify the forms displayed to users during client events. This separation also permits line rules and succeeding rules to be generated automatically, reducing the development burden of keeping these in synch. This offers a powerful and dynamic way to design processes that is different from K2.net 2003. Beyond not needing to hardcode potential actions in the form, separating Actions from Outcomes also allows rights to be associated with Actions, so that at runtime these rights are queried and only the Actions that a user is given permissions to perform are displayed on the form. This can be controlled using the K2 Workspace Management Console as well as when worklist items are delegate to another user.

SMARTOBJECTS

SmartObjects and their associated SmartObject Services layer allow business data to flow into and back out of processes. There are several SmartObject Services that are installed with K2 blackpearl, including:

- Microsoft SQL
- SQL Reporting Services
- Salesforce.com
- Active Directory
- Workflow
- SharePoint

Custom SmartObject Services can also be written to connect to any line of business (LOB) system to query and update data from those systems through SmartObjects. SmartObjects not only allow data from LOB systems to be incorporated into a process, but also allow data captured within a process to be more easily repurposed for other processes and systems using the .NET Data Provider for K2 SmartObjects, an ADO.NET adapter that can be used to query and update SmartObject data from any .NET client. Abstracting data from the process also allows the process data load to be smaller, decreasing the SQL requirements for the main K2 Server tables and increasing the performance and scalability of your processes.

K2 BLACKPEARL INSTALLATION, CONFIGURATION AND SECURITY BEST PRACTICES

Your first point of reference for installing K2 blackpearl should be the Getting Started Guide, which is one of the three main content Help files that ships with K2 blackpearl. Also reference the [K2 blackpearl Deployment Planning Guide](#) located on [K2Underground](#) for detailed information regarding supported topologies for installation and planning.

There are a few other best practices to keep in mind regarding your installation, configuration and maintenance of your K2 server.

- **Always install the latest release or service pack.** This helps ensure that you have the latest feature and functionality enhancements as well as the latest bug fixes. The latest release is always available at <https://portal.k2.com/downloads/bp/Default.aspx>. Pay particular attention to the release notes of the latest release for information regarding the specific enhancements and fixes included in the release. Also take note of the [K2 Compatibility Matrix](http://help.k2.com/en/blackpearlmatrix.aspx) (<http://help.k2.com/en/blackpearlmatrix.aspx>), for the latest information on K2 blackpearl compatibility with supporting software versions and platforms. Before applying any service pack or installing the latest release, you should complete a full testing cycle with the update in your QA/Test environment.
- **Only install patches/hotfixes that need to be installed.** If you are not experiencing a problem with your K2 installation or processes currently running, you should not install any patches/hotfixes. They are only meant to be installed by customers who are experiencing the specific issue that the patch/hotfix addresses. Updates, which should be installed in a test environment first and then followed in a production environment, are made available on a regular basis and include the latest patches/hotfixes in a single installer.
- **Only use physical Host (A) records in your domain.** Using DNS Alias (CNAME) records may be tempting because they help with scalability, Identification, Disaster Recovery and troubleshooting, but there is a known Kerberos delegation problem between an Internet Explorer client and a Windows server that resolves CNAME records to Host A records, causing authentication to fail (see [Microsoft KB911149](#)).
 - **Use Host (A) records like aliases:** Use a Host (A) record that points to your active Web server. The web server should be configured to use a host header that matches your service principle name (SPN) Kerberos record. During failover and maintenance scenarios, you update your Host (A) record to point to the IP address of your standby server. This precludes any Active Directory changes and subsequent replication wait times, and has the added benefit of not invalidating active Kerberos tickets.
 - **Note:** If you install the hotfix for the DNA Alias (CNAME) issue or if Microsoft makes other changes to the way Windows clients authenticate with Windows servers, other best practices may be developed for K2 servers using Kerberos authentication.
- **Environment Library template naming.** For more information about Environment Library naming conventions and best practices, see the upcoming K2 blackpearl Professional book and [The K2 blackpearl Environment Library](#) whitepaper on [K2underground.com](#) (http://k2underground.com/files/folders/technical_product_documents/entry27110.aspx).
- **Setup Development and QA/Test environments.** Establishing these environments at the same time as designing your Production environment allows you to determine your hardware, licensing, testing and troubleshooting plans much better than allowing these things to evolve on an as-needed basis. Your QA/Test environment should resemble your Production environment as much as possible.
- **Use MSBuild packages for deploying processes and SmartObjects to QA/Test and Production (non-Dev) environments.** This is typically required in enterprise environments due to the responsibilities

of keeping these environments running, but it is also a best practice to deploy using MSBuild to minimize the overhead on these servers. For more information see [KB000188 - How to use the Deploy Package \(http://help.k2.com/kb000188.aspx\)](http://help.k2.com/kb000188.aspx).

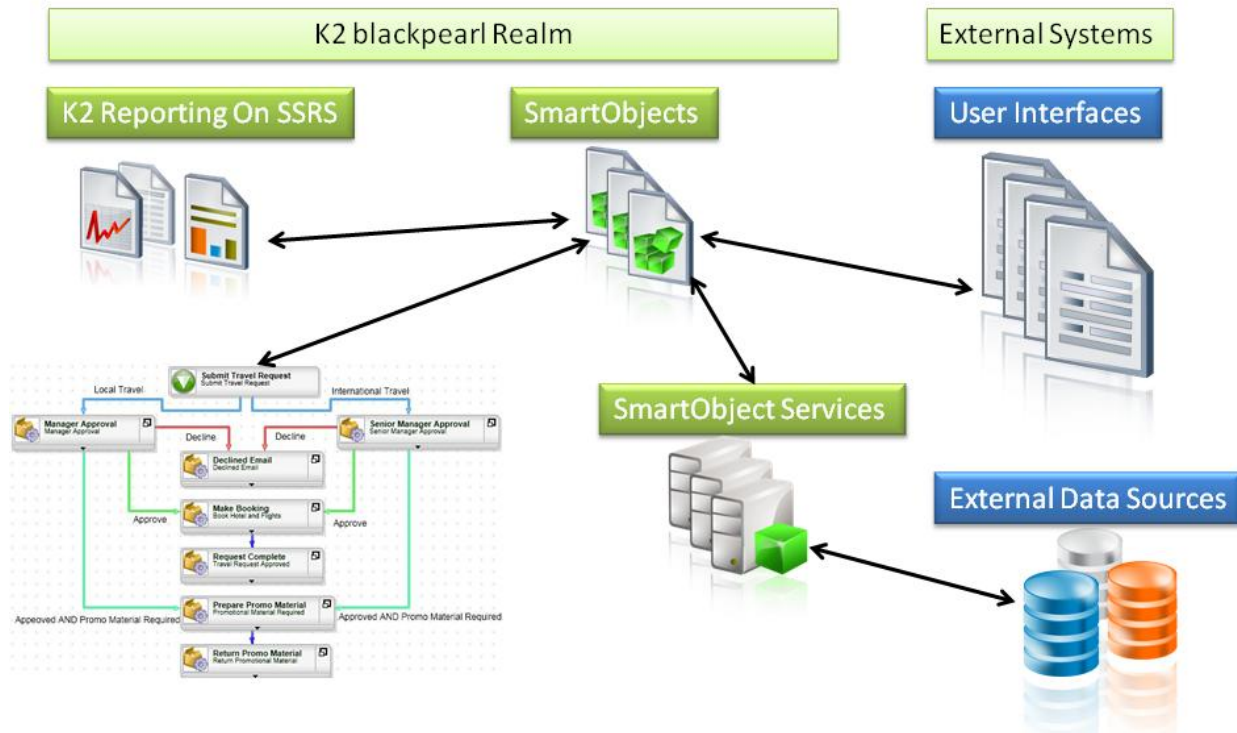
- **Ensure other dependencies are in place for each environment.** Make sure other dependencies, such as custom service objects, Web forms and Web services are deployed and functioning as well. For error-repairing processes in a production environment, have a workstation available with the K2 Designer for Visual Studio installed on it to repair process errors.
- **Assign server rights.** As soon as you install your K2 server and verify that it is properly configured, you should lock it down by assigning rights to the server. Rights control who can deploy processes and SmartObjects to the server, and who can see items such as the Report Designer, the Notifications and Custom Event designer, and the Management Console in K2 Workspace. This is done using the options in K2 Workspace and the Management Console and it is important to remember that in K2, access is typically not restricted until permissions have been assigned. Once they are assigned, those not specifically granted some level of permission will not have access to the administrative areas of K2 Workspace.
- **Only assign K2 Server rights to accounts which require such rights.**
 - **Admin** – Account will have full administrative rights on the K2 server
 - **Export** – Account has only rights to export (deploy) to the K2 Server
 - **Impersonate** – Account has internal impersonation rights within the K2 workflow context, not to be confused with Kerberos impersonation. Only assign this right for accounts that execute code requiring impersonation rights within K2. **Use of Kerberos impersonation over K2 impersonation is highly recommended as it is more secure, scalable and K2 does not act as an authentication mechanism.**
- **Assign process rights to groups instead of users.** Assigning rights to groups allows greater flexibility and maintainability of your K2 server. It may be necessary to assign Start rights, for example, to individual users, but try to avoid this when possible. Assigning rights to groups at the server level is not possible as of this writing but is being investigated.
- **Refrain from using K2 management APIs** within process solutions as the use of these requires that the identity of the user executing the code has administration rights. In particular this includes the management APIs contained within the SourceCode.Workflow.Management, SourceCode.ManagementAPI, SourceCode.SmartObjects.Services.Management, and SourceCode.SmartObjects.Services.SmartBox.Management assemblies, but any assembly with 'Management' in the name typically requires permissions on the server that a typical user will not have. Occasionally use of these APIs is required but it should be kept to a minimum.
- **Refrain from using the K2 workflow API** in a process, in particular to access the current process instance. Trying to use the SourceCode.Workflow.Client API on the same process instance from which it is being invoked can cause some stability issues due to the way the K2 server handles process execution. You should also refrain from using the API to work on different process instances unless there are no other design options and you are absolutely certain that it will not interact with the process instance from which you are making the API call.
- **Always make a backup of all K2 databases.** Upon completion of successful configuration, a base-level backup of your databases, configuration files and any deployed custom service objects will aid in recovery.
- **For Kerberos troubleshooting, see the Getting Started Guide.** There is a wealth of information in this guide for troubleshooting potential problems with Kerberos. Also see the [Security and Kerberos Authentication with K2 Servers](#) and [K2 and Multi-Tier Environments](#) whitepapers on K2underground.com

(http://k2underground.com/files/folders/technical_product_documents/entry21001.aspx) and consider the following facts:

- **User Base** – Where will users be coming from? Within the network, over VPN, from the Internet or a three? Will all these users have accounts in Active Directory or is another user store required?
- **Domains** – How many domains are there? Where will the K2 server reside and which domains will it interact with?
- **Trusts** – What trusts are existent between the identified domains? Are there any one-way trusts which will block K2 Active Directory User Manager (ADUM) from carrying out LDAP queries?
- **Forests** – If there are there multiple forests involved, is there forest trust to allow Kerberos tickets? Two-way forest trusts are usually required but it depends on where users and other required resources are located. Contact K2 support if you do not have a two-way trust established and your K2 server cannot authenticate some users.
- **Functional Levels** – What level do the domains and forests run at? Do you require any of the features provided by a native Windows 2003 level?

SMARTOBJECT DESIGN BEST PRACTICES

Ideally in K2 blackpearl, the focus for process design is centered on a business object approach. All items in the process are evaluated on how they interact with these business objects, in this case SmartObjects, and what the data represented by these business objects means to the business itself. Where the process intersects the business data should become evident during the process discovery phase of the project, and will represent the critical results of processes activities and events.



An individual process is broken down into its smallest business components. These components are then used as the data source of the following components:

- All rules of the process
- User Interfaces
- Reports

The SmartObjects are then mapped to their appropriate backend systems via SmartObject Services. This allows changes to the back-end system at any point in time without affecting the business application. When using the SmartObject Services shipped with the product, no code needs to be written for read and write access to backend systems. Custom SmartObject Services, often referred to as Service Objects, can also be written to talk to various backend systems that do not have a corresponding SmartObject Service that ships with K2 blackpearl.

Note: The Active Directory service objects that ship with K2 blackpearl 0807 do not allow updates to Active Directory. There is a service object available on K2 blackmarket that allows updates (<http://k2underground.com/k2/ProjectHome.aspx?ProjectID=3>).

Using SmartObjects in this manner allows a process to store nothing but a reference to a SmartObject ID in the process data fields, which abstracts data from the process and offloads work from the K2 workflow server. Using this one data field, a SmartObject Reference can be made in the process definition. This SmartObject reference becomes a re-useable object in the process definition which, in turn, is employed in line rules, start rules, escalation rules, destinations and the like. As a benefit of abstracting data from the process, the process becomes

more lightweight in terms of instance data handled by the server and database storage needs as well as permitting this business data to be more easily leveraged outside of the process.

Follow your company's naming conventions when defining properties and methods in your service objects, and provide understandable descriptions for service type properties and methods, since these are surfaced to users that need to design SmartObjects that consume these methods and properties.

SmartObjects are developed once the underlying service types have been registered on the K2 server and an instance of the service type created. This instance registration is accomplished through the K2 Workspace, and can also be accomplished with the BrokerManagement.exe tool that allows you to supply a specific service instance GUID for each custom service type.

You might choose to keep all SmartObjects within separate solutions based on the service types, but since SmartObjects do not necessarily have to be bound to a specific service type, this approach may not be logical. A single solution for all SmartObjects may be a better approach, as you can ensure consistency and avoid naming conflicts, duplicated and overwritten SmartObjects, and to centralize the deployment of SmartObjects. Where this is not possible, develop groups of SmartObjects, each within a single solution, using an approach that makes sense to your business, but try to avoid having any single SmartObject contained within multiple groups. A hybrid approach that may work in your environment is to develop all business-related, common SmartObjects in a single solution, and have separate solutions for application-specific SmartObjects.

With the single solution approach, developers are able to work on individual SmartObjects at the same time, although deployment requires them to 'exclude' SmartObjects they are not currently designing.

Note: When deploying SmartObjects, K2 will automatically create a Category for the project name and sub-categories for each of the folders in the project. By including all SmartObjects in a central solution, you can ensure that SmartObjects are logically grouped, named and deployed to the K2 category system.

There are two more important things to remember when designing SmartObjects.

- **K2 provides two mechanisms to retrieve data from SmartObjects:** The.NET Data Provider for SmartObjects (a standard ADO.NET data provider) and the SourceCode.SmartObject.Client API. Each approach provides different benefits and requires a slightly different way of coding, but the important thing to keep in mind is that because data can be exposed through SmartObjects to other non-K2 applications, the SmartObject infrastructure may begin to play a larger role within your business than initially planned for. However, keep the following point in mind as you create SmartObjects.
- **Use SmartObjects as a data abstraction layer instead of your Enterprise Data Model.** With the first bullet here in mind, it may be tempting to use SmartObjects as your Enterprise Data Model to abstract and hide legacy aspects of your business data, but that is not considered a best practice. SmartObjects provide a way for processes and other applications to intelligently use data about your business for business purposes, and does not supplant the need for a proper data model. In many cases, SmartObjects are built upon the Enterprise Data Model so they fit into existing conventions yet still ease K2 process development.

If the business object centric approach does not work for your business, or for every process that you design, there are other approaches that can work equally well depending on the project requirements. Here are the three main approaches and some Pros and Cons of each, starting with the business object centric approach:

BUSINESS OBJECT CENTRIC

PROS

- Scalable
- Reporting is easier
- Data is accessible outside of the process

CONS

- No automatic auditing
- InfoPath secondary data sources (based on SmartObject queries) are not validated, so if you need to validate data you must create InfoPath rules to copy data from the secondary data source to the main data source, or include your SmartObject queries in the main data source as an option when adding them to the InfoPath form.

PROCESS CENTRIC (MANY PROCESS DATA FIELDS)

PROS

- Enables granular auditing

CONS

- Potential scalability issues
- Difficulty in reporting
- Data locked up in process

INFOPATH CENTRIC (EVERYTHING IS IN THE INFOPATH XML FIELD)

PROS

- Eases InfoPath form development

CONS

- Potential scalability (many copies of the InfoPath XML)
- Only XML document level auditing
- Data level reporting is limited
- Split and merge (parallel plans) requires SmartObjects

PROCESS SELECTION

When looking at process automation, just about any process in a business can be automated. But the central question is will it be worth the effort? Here are some guidelines for identifying processes that, through automation, can deliver a good return on investment.

The more questions you answer "Yes" to across the following three categories, the more likely the process will benefit your business when automated. This list is meant to get you thinking about your business and the types of processes you have. It is not meant to be an exhaustive list of questions that you will have to find answers to once you choose a process to automate. Look at the other sections in this whitepaper for the kind of information you will need to gather once you start the process analysis and design phase.

PROCESS QUESTIONS

1. Does the process require reporting, auditing, compliance or version control?
2. Can the process be mapped? Or has the process been mapped and optimized already?
3. Is this process in operation today, electronically, in paper or otherwise?
4. If the business process is new to the business, have the necessary participants of the process been informed and involved in the change management and business improvement exercise?
5. Do you have a paper forms and folders that are passed around?
6. Is this process time sensitive? Are there service level agreements? Do you need escalation points when specific time limits are reached based on your SLAs?
7. Do you have a business process that requires more than one type of review or decision at the same time?
8. Does the process share information with any other process?
9. Does the process run for a long time?

PEOPLE QUESTIONS

1. Will tasks need to be performed on behalf of another person?
2. Will the process involve the participation of the entire organization?
3. Are the process participants geographically dispersed?
4. Does the process need to escalate if someone fails to action it?
5. Do multiple people need to review and sign this off?
6. Are external parties involved in the process, for example regulatory agencies, customers and partners?
7. Is it important to notify users of process status?
8. Is overall process visibility important, for example for employees, managers, executives or auditors?
9. Do you need visibility into your process, like who is doing what and how long does tasks take to complete?

INFRASTRUCTURE QUESTIONS

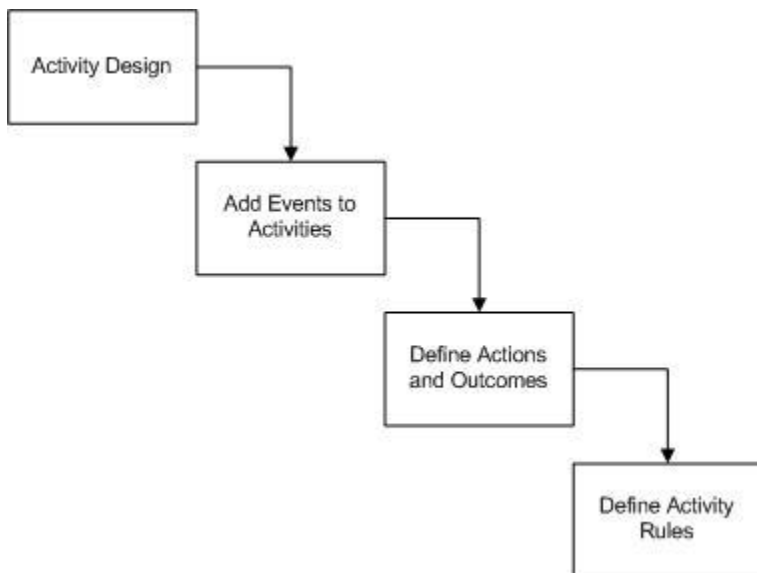
1. Does your company have the infrastructure to automate this process?
2. Do all individuals involved in the process have access to a computer?

3. Do you need to obtain information from other computer systems? Are these systems able to integrate with other systems through APIs or Web services? Does the vendor of the software support and allow this integration to be performed?
4. What, if any, are the licensing implications of this 3rd party application of allowing this integration? Has this cost been accounted for in the total cost to automate?
5. Do you have disparate systems that need to participate within a business process?
6. Are these systems owned and hosted in-house or are they systems used and maintained by external parties?

Process automation can bring many benefits to a business, and there are K2 features that can solve some of the more difficult problems of non-automated process, like version control, tracking, compliance, auditing, multiple or parallel approvals, escalations, delegation, surfacing business information from other systems, and increasing the accountability and transparency of your business operations.

PROCESS DESIGN BEST PRACTICES

There are many approaches to designing K2 blackpearl processes, but many of them require what is referred to as the Waterfall methodology. You start by creating a high level activity design and continue by adding more and more detail into the process. Once the design is complete, the actual creation of the process follows.



The more detail that is placed in your design, the more complex the actual design environment will be. A simple design could be done using the K2 Web Designer for SharePoint, where a more complex design should be done using the K2 Designer for Visual Studio or the K2 Designer for Visio 2007.

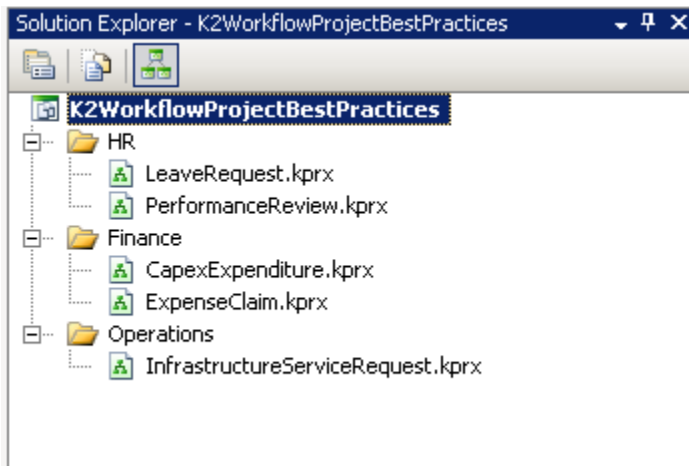
PROJECT STRUCTURE

As mentioned in the SmartObjects section, you should separate your SmartObject projects from your Process projects. The drawback of having multiple Processes or SmartObjects in the same project is that, by default, when deploying the project to the K2 server, all processes and SmartObjects in the project are deployed. Should you decide to exclude certain items from the deployment, they need to be excluded from the project build task (right click the project and select 'Exclude from build') before deploying.

Depending on the scope of your processes, separate projects should be created for different processes. The deciding factor is usually whether different development teams will be working on the projects simultaneously. If they will, separate projects are required for each team.

Generally, projects are subdivided into folders that cover specific processes. For example, suppose there are a set of processes specific to HR, Finance and Operations. One would create separate folders for each of these areas of the business and create the processes for that business area in the relevant sub-folder. The folder structure is deployed as part of the process title to the workflow server. A well-organized structure can allow users to find relevant processes more easily than other approaches.

Workflow projects will be deployed using the project name as the top-level folder, then any sub-folder and finally the process name. For example, the following project structure is deployed to the server and results in processes that reside in separate folders, as shown in the subsequent figure.



Process Name	Folder
InfrastructureServiceRequest	K2WorkflowProjectBestPractices\Operations
LeaveRequest	K2WorkflowProjectBestPractices\HR
PerformanceReview	K2WorkflowProjectBestPractices\HR
CapexExpenditure	K2WorkflowProjectBestPractices\Finance
ExpenseClaim	K2WorkflowProjectBestPractices\Finance

By grouping processes according to business unit or functional area, users can group and search workflows more easily. The name of the workflow is typically presented to the user as <Folder Name>\<Process Name>, so be descriptive but at the same time limit the number of combined characters used in project, folder and process names so that it is easier for users to see the entire name.

NAMING K2 ITEMS

It is important to choose the right names for each K2 item, including your projects, folders, processes, activities and events. Choose something descriptive but not too lengthy. Name your activities using nouns and your events using verbs. For example, Manager Approval is a good activity name, and Approve Request is a good name for the client event within the activity. All of these names are used at some point in the management of the process as well as in the reporting of the process, so choosing good names is important and considered a best practice.

The use of Pascal Casing (i.e. PascalCasing) or Camel Casing (i.e. camelCasing) is recommended. Do not use Hungarian notation to name K2 items. In earlier days most programmers liked it - having the data type as a prefix for the variable name was very useful. This practice however, has fallen out of favor and is not recommended in .NET coding standards. The use of meaningful, descriptive words to name variables is used to describe .NET variables, and standards state that variables should use camel casing and methods and classes use Pascal casing. Naming K2 items other than the custom code employed and data variables should probably follow Pascal casing, but this is more of a preference. Using one of these methods is a best practice.

GRAPHICAL DESIGN RECOMMENDATIONS

In this section you will find information about process design elements instead of client event form design best practices.

In large processes where many activities are defined, it becomes helpful to color code the activities based on the task that is performed in that activity. The color legend can be defined for the particular process needs, but as an example use something similar to the following.

- Blue = Client Event
- Yellow = IPC Event
- Grey = Web service or Integration call
- Red = Process End point
- Green = Process Start

It is recommended to display lines that result in parallel processing using a standard color, such as purple. In addition, it can be beneficial to color the lines according to their rules. The following can serve as a general guideline but depending on the actual process more colors can be added.

- Green = Approved
- Red = Declined
- Blue = Send for Rework

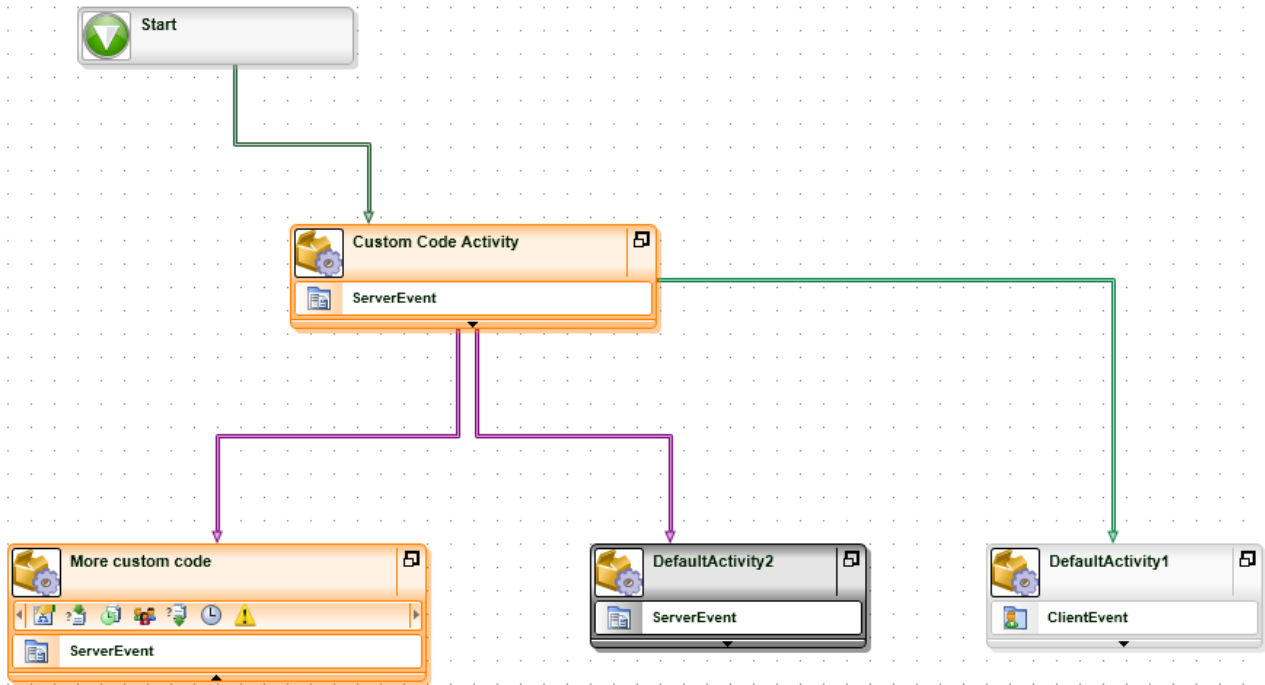
Note: These colors will not be seen by end users in the process View Flow.

Lastly, you should color events or activities that contain custom code using a reserved color, such as orange. This helps developers understand which activities and events can be updated to newer templates, and which should not. The 'Update Design Templates' function will overwrite any customization applied to the default wizards with the exception of the Default Server Events (code and WF). If the code behind an activity or event is modified updating the design template removes those modifications and updates to the newer template. If customized activities are in a different color, such as orange, developers can easily see which activities have been customized and exclude these from updating.

Note: Processes developed with an earlier version of K2 blackpearl should always be updated to the latest template versions at some point. Feature and functionality changes in the newer version may cause problems when coupled with the older version of the template. It is considered a best practice to update the design templates with each release if the process is to be redeployed.

Another related best practice is to avoid code customizations outside of a default server event.

In the following figure, activities that are performed by systems are colored black, activities performed by humans are left in the default color, and activities with custom code are colored orange.



SOURCE CONTROL

With the release of K2 blackpearl 0807, using a source control system is much better. In previous releases, the Workflow Foundation (WF) extender projects were temporarily written to the project path, so upon check-in the extender projects were also checked in. They are not necessary to check-in, however, and will cause problems

because they are treated as temporary in nature by K2. In K2 blackpearl 0807, the extender projects are no longer written to the project path but rather to the temporary folder, and are therefore not checked-in.

Developers who are working in K2 projects should be using the same local source control path as every other developer. This prevents any problems with path length issues in the actual project files if the length starts approaching the Windows path length limit, and it results in greater predictability. The common path should be something short and understandable, such as "C:\K2Proj".

Note that two developers cannot work on the same file in a project at the same time, since changes are saved into the source file during development. It is not recommended to merge changes to these files using source control tools, since the internal file structure of the .kprx is both proprietary and complex. There is a very real chance of file corruption should the files be manually edited. Although two developers will not be able to edit the same process file (.kprx) or SmartObject file (.sodx) simultaneously, they will be able to develop on the same project but in different files simultaneously.

PROCESS DATA

As described in the SmartObjects Design Best Practices section, storing data using SmartObjects is the best practice, but in some cases you will want to store data in the process itself. In all cases it is important to understand the differences between the various data fields available in K2 blackpearl.

Data fields allow a process to store metadata, either for the entire lifetime of a process or for the duration of a single activity.

PROCESS DATA FIELDS

Process data fields exist for the entire lifetime and their scope is global to the process. Data fields should only contain data that are tightly related to the particular business process or activity that they are defined for. Ideally data fields will only contain pointers or ID values to the business entities and data that is stored in the application databases.

The process should almost never contain data fields that will be displayed in the UI layer. The only exception to this is when worklist filtering is required on certain fields outside of the standard K2 workflow fields. For example if the worklist needs to be filtered by department, then it would make sense to have a Department data field in the process and display this value in the custom worklist. In this case, consideration must be given to the size of a user's worklist. As the worklist size and number of data fields increases, filtering reduces performance of the worklist.

ACTIVITY DATA FIELDS

Activity data fields exist for the lifetime of the activity instance and have a scope of the activity instance. In K2.net 2003 activity data fields are duplicated for each destination user assigned to the activity. For this reason these data fields have the tendency to easily grow to enormous proportions, consuming large amounts of server memory, network bandwidth, and possibly also disk storage. Irresponsible use of activity data fields with large amounts of data assigned to many users is strongly discouraged. In K2 blackpearl the default setting for activity

destination rules are to Plan just Once, and by default the data fields will not be duplicated in the same way. However, it is still possible for a process developer to change the destination rule to plan per destination, and in such cases data fields will be duplicated for each destination user.

XML DATA FIELDS AND INFOPATH FORMS

XML data fields are used to store much data about a process, such as the SharePoint integration data, workflow history and the actual InfoPath forms for those types of processes. Be careful when using the InfoPath file attachment control because this data is serialized and added to the InfoPath XML file, and then copied to the XML data fields. A better practice is to use something like a SmartObject to upload a document to a MOSS library rather than adding it directly to the InfoPath form and potentially causing database bloat from the increased XML data size. This is particularly important on activities that are Plan per Destination as the InfoPath XML is copied once per destination and slot, which can increase the load on the server significantly. It also causes the problem of splitting and merging the XML data from the InfoPath form, which can be avoided using SmartObjects. Where many users could potentially be uploading large files simultaneously, use performance measures to monitor the server performance. If it becomes too much for the K2 server to handle you may need to write a custom Web service to handle large, simultaneous file uploads.

DATA ON DEMAND

The use of On Demand setting on data fields, as shown in the following figure, is a very important feature that is often neglected by developers. If On Demand is not enabled for data fields (by default it is enabled) it means that every query to a K2 Worklist Item retrieves every piece of process data, which can put a strain on the SQL Server Database in high-load environments where many K2 processes are simultaneously running. Good design considers each data field individually to determine if On Demand is necessary. Certain data fields that are accessed often might make the process perform faster if On Demand is switched off so that the data field operates almost in a cached manner.

For more information about Data on Demand, see the K2.net 2003 article [KB000102 - K2.net 2003 – Data on Demand explained](http://help.k2.com/kb000102.aspx) (<http://help.k2.com/kb000102.aspx>). The functionality is the same in K2 blackpearl.

Add Data Field
✕

Name:

Data Type: ▼

Category: ▼

Initial Value:

Metadata:

Description:

Hidden
 Keep Log
 On Demand
 Keep Audit

DATA AUDITING

Auditing of data fields is an important feature which allows keeping track of all changes to data and XML fields and who made the changes. This function, however, can create extra server load as well as growth in database size because all changes are recorded and stored in the DB. For this reason the use of the Keep Audit setting on data fields must be carefully considered on an individual basis for each data field. As a rule of thumb, only data fields that are being updated by humans or external systems need to be audited. Data fields that are always updated by the server itself will only need auditing if there is a requirement to see previous values but it will have no use in determining who affected the change.

For more information about auditing, see [KB000298 - K2 Auditing and Logging](http://help.k2.com/kb000298.aspx) (<http://help.k2.com/kb000298.aspx>).

Note: Data auditing only applies to data and XML fields and not to data contained within SmartObjects.

DATA LOGGING

All data is logged by default. This means that any change that occurs to the state of the process instance within the transaction database (K2Server) is automatically logged to the reporting database (K2ServerLog). If you uncheck the Keep Log option as shown in the figure above, the data from the field will not be copied to the K2ServerLog database. This database is used primarily for reporting purposes.

If you need to keep large amounts of data in process data fields, it can be beneficial to turn off logging for this data so that the overhead on the K2ServerLog database is kept to a minimum and data is not stored that may not be relevant to reporting on the process. For example, if a file is passed into the process data field, there is no real value in having that same file stored in the K2ServerLog database since it doesn't represent data commonly used in reporting.

RULES

This section will outline some information detail and best practices about many different types of rules you can modify in K2 blackpearl processes.

PRECEDING VS. START RULES (MAKE SURE YOU KNOW THE ORDER AND WHAT THEY'RE USED FOR)

A preceding rule is not a 'wait until this is true' type of rule. Every time a specific line is followed, the preceding rule is evaluated once, and if it returns true, then that activity will start. The only remaining question is when it will start, and that then gets determined by the Start rule. Another way to think of this is that a Line Rule, when evaluating to True, leads to an activity. The activity then executes the Preceding Rule, which is used to evaluate data to determine *if* the activity should continue. If Yes, then the Start Rule fires which contains a time-based logic to determine *when* the activity should begin. The process is dehydrated and another thread is responsible for rehydrating the process at the appropriate time according to the Start Rule calculation, which takes into account the configured Working Hours.

It is important to remember that a non-configured (empty) rule will always evaluate to True, so when you do not have data-based or time-based requirements for starting an activity, the activity will automatically start when the line leading to the activity is complete.

The order of rules is:

- Line Rule
- Preceding Rule
- Start Rule
- Destination Rule
- {all events}
- Succeeding Rule

LINE RULES (TRY TO KEEP MAINTENANCE LOW)

In K2 blackpearl the concept of Actions and Outcomes is used. All client events have at least one action and one outcome. Each action represents a decision that the user can make at that point in the process. The outcome represents the process route taken based on the action taken. In most cases there will be a one to one correlation between actions and outcomes; however this does not always have to be the case. Outcomes can also be

combined with standard line rules, or in other words a line can be configured to execute when a user selected the Approved action and the approval amount is greater than 5000. However, it is recommended that lines that contained modified rules are kept to a minimum. This allows greater process maintainability because if the wizard is re-executed and the lines regenerated, the modifications are lost. The option to automatically generate lines based on actions is off by default once lines have been generated the first time, but this is something to be aware of.

Be careful not to design looping line rules. Don't create a polling activity that loops back on itself every few minutes to check if a certain condition on an external system has been met. Building this type of functionality into a K2 process can add significant load on the K2 server and databases. If you need to do something like this use an asynchronous server event as described below.

ESCALATIONS (NOTHING IN EXCESS)

Escalations are used to escalate a particular item if it has not been actioned by a user after a certain amount of time, or at predefined date and time. Escalations should not be used to perform standard tasks that you want to execute in the activity. For example, they should not be used to send standard emails one second after the activity was started. Generally, if you have for than 50K to 60K records in the _Esc table of the K2Server database, you may start running into scenarios where escalations will not fire as expected.

In K2 blackpearl there are three different levels of escalations.

- Process
- Activity
- Event

Note: Only activity escalations were available in K2.net 2003.

Activity escalations are the most common, but the other escalation levels can be used when track different durations. For instance, use a Process escalation to track the duration of the entire process, an Activity escalation to track the duration of a specific instance while ignoring individual users in a multi-slot instance, and an Event escalation when tracking a specific event a tied to a slot that may have some user context when the plan per destination activity plan is used.

DESTINATION RULES (KNOW THY SELF)

With destination rules you may find it difficult to understand the implications of the advanced options when the process gets to your main activity. If you need to do anything other than the default Plan just Once activity plan, or you are using dynamic roles, you should read the whitepaper K2 blackpearl Roles and Advanced Destination Rules (http://www.k2underground.com/files/folders/technical_product_documents/entry20948.aspx) carefully and then fully-test your process in a Development or QA/Test environment with users and roles that represent your actual process.

Abstracting your user management by using roles and, in some cases, SmartObject queries, allow you to manage current and future instances of a process without redeploying it. Also be aware of functionality that is disabled

when using the default Plan just Once plan, as detailed in the blog post [Sending emails to Destination users \(http://k2underground.com/blogs/blackbelt/archive/2008/06/30/sending-emails-to-destination-user.aspx\)](http://k2underground.com/blogs/blackbelt/archive/2008/06/30/sending-emails-to-destination-user.aspx).

Be aware that if your destination users are not specified using a fully-qualified name (FQN), such as DomainName\UserName, the default user manager security label is pre-pended to the name when the destinations are resolved. The default security label is "K2" and corresponds to Active Directory (AD) users. An example of a user in the DENALLIX domain is "K2:DENALLIX\Mike". Other user managers, such as the SQL user manager, use different security labels. This means that if you assign tasks to SQL users using their unqualified user names, the server prepends "K2" to the SQL user and assigns tasks to users that don't exist.

Also be aware that when using an InfoPath form in a client event, and you are using a Plan Per Destination -- All at Once (parallel) plan or you have configured multiple slots, there is potential for data to be overwritten. To avoid this you must either use SmartObjects to store each user's data or update the InfoPath client event code to only update separate fields for each user.

SUCCEEDING RULES AND ACTIVITY SCOPING

Based on the way in which you plan your activities, and especially if you do not use the Plan just Once default plan, you will need to determine where to put server events and activity data manipulation events. For example, if you have a client event in which users updates 10 data fields and you want to push that data immediately to a SmartObject, you cannot put the SmartObject event inside the same activity. That data updated by the user isn't available as part of the larger process until the activity completes, which happens after the succeeding rule is applied. In this case, the server and data manipulation events should always be placed in a follow up activity.

In contrast, in a parallel activity plan (Plan Per Destination, All at Once), you may want to capture information about actions that each destination user takes before the succeeding rule is applied. In this scenario, call the appropriate SmartObject method (typically either Create or Update) immediately upon submit to send the information to a SmartObject. This gives you the flexibility to query all activity data because it is abstracted from the process, and data about each destination user is kept in a separate record for easier merging later on, such as in a server event in a subsequent activity.

Note: If you have more slots than destination users, your succeeding rule will never fire and your activity will appear "stuck" because it will not advance to the next activity. Ensure that you have at least as many destination users as you have slots.

USING GOTOS

Using GoTos, including using the GoToActivity API method, in a process should be used with care as it will expire all currently active Activity instances within the process instance and jump the execution of the process to the specified target activity. This can lead to lost worklist items, premature process completion and orphaned IPC child processes, as well as to the loss of reporting visibility. This can also affect the View Flow report, where lines will not appear as Red or Green since they were not followed. Sometimes GoTos are necessary but other times a spider workflow is a better approach. For more information about designing spider workflows, see [Workflow Design - Spider Workflow \(http://k2underground.com/blogs/adriaanandolaf/archive/2007/06/27/workflow-design-spider-workflow.aspx\)](http://k2underground.com/blogs/adriaanandolaf/archive/2007/06/27/workflow-design-spider-workflow.aspx).

PROCESS VERSIONING

While the K2 server maintains versions of your processes automatically, it can prove useful to place a version number in your start activity or process data field, as this can prove useful when looking at the global worklist or View Flow data fields, and can help you easily identify the version of the process a running instance is using.

PROCESS SIZING

It is recommended to limit the number of activities in a process as much as possible. If a process has many (generally more than 20) activities, it is usually a good indication that the process can benefit from a business process reengineering (BPR) exercise, or that the process should be subdivided into smaller processes with IPC calls between the sub-processes. Large processes not only complicate development and maintenance but also reduce re-usability and insight into the process flow. However, be careful to understand how splitting a process into multiple pieces may affect reporting data. It is easier to generate a report on a single process, even when it is very large. You will have to weigh the benefits of reporting to the business and the benefits of manageability when making this decision.

Also pay attention to the actual size of your process, mainly the process (.kprx) file. Especially if you are using a source control solution, when this project file grows beyond about 20MB in size, it may become unwieldy and benefit from being split into multiple parts. This number should not be thought of as a rule but rather a guideline. Some processes that are over 10MB in size may also benefit.

You should choose a process architecture that is not overly complex. A process containing hundreds of data-intensive activities, such as InfoPath and SharePoint, or that have large data or file attachments, is not recommended. Abstracting data and logic into external assemblies, external referenced data, service objects and SmartObjects, and IPCs is a better approach that makes it easier and more predictable to design, deploy and maintain.

USING INTER PROCESS COMMUNICATION EVENTS

In the case of an IPC event call, if you want the parent process to wait for the child IPC, then you must call the IPC 'synchronously', in which case the parent will wait until the child completes and optionally you can pass back data fields from the child to the parent.

The only other mechanism (other than IPC) for making a process 'wait' until certain external conditions are met is an asynchronous server event. In that scenario you could call an external system, such as BizTalk, and pass in the serial number and tell the server event to wait. When the external system is done, it can make a call back to K2, re-instantiate that event instance using the serial number, and finish the event (optionally passing in data) so that the process continues.

How does an asynchronous server event differ from the normal synchronous code event? When the workflow engine encounters a *synchronous* server event it executes the event and then continues the workflow. In an *asynchronous* event, the workflow engine executes the event and then stops. When you first drag a server code event onto the workflow design canvas, by default it is synchronous. A server code event can be changed from synchronous to asynchronous by adding one line of code to the event:

```
K2.Synchronous = false;
```

To notify the workflow the event is complete a key piece of information must be made available to the external system: the serial number of the activity containing the asynchronous server event. Every activity (whether you use it or not) has a serial number that uniquely identifies it. A serial number is similar to the correlation ID in used in message queuing. This serial number allows you to use the K2 API to open the correct server item and complete the event using the **Finish** method of **ServerItem**:

```
public static void CompleteServerItem(string K2Server, string serialNumber)
{
    if (string.IsNullOrEmpty(serialNumber)) return;
    SourceCode.Workflow.Client.Connection K2Connection = null;

    try
    {
        SourceCode.Workflow.Client.ConnectionSetup K2Setup = new ConnectionSetup();
        K2Setup.ConnectionString = K2Server;
        K2Connection = new Connection();
        K2Connection.Open();
        ServerItem serverItem = K2Connection.OpenServerItem(serialNumber);
        if (serverItem != null) serverItem.Finish();
    }
    catch(Exception ex)
    {
        myLogException(ex);
    }
    finally
    {
        if (K2Connection != null)
        {
            try
            {
```



```

        K2Connection.Close();

        K2Connection.Dispose();

    }

    catch(Exception ex)

    {

        myLogException(ex);

    }

    finally

    {

        K2Connection.Close();

        K2Connection.Dispose();

    }

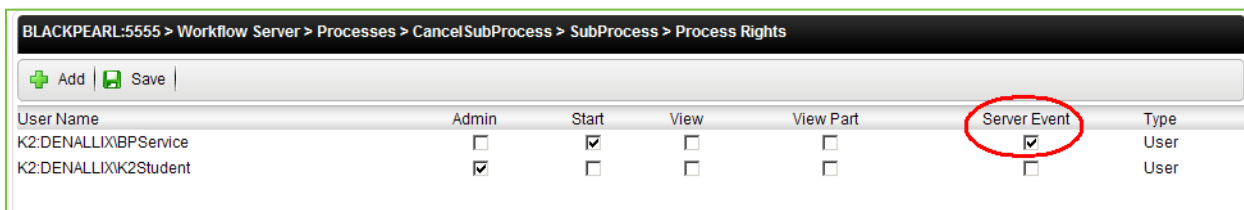
}

}

}

```

You can also use the ServerItem class to set process and activity level fields to transfer information back into the workflow. This code could be invoked from a .NET assembly, a web service, or whatever approach best matches your situation; just add a reference to the **SourceCode.Workflow.Client** assembly. The identity used to connect to the K2 Server needs be assigned the **Server Event** right for the process in the Management Console.



User Name	Admin	Start	View	View Part	Server Event	Type
K2: DENALLIXIBPService	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	User
K2: DENALLIXIK2Student	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	User

A common design scenario for workflows is the need to be able to cancel asynchronous IPC processes at any time in response to some action in the main workflow. This scenario typically occurs when the originator decides to cancel a request that has started several independent sub processes to perform part of the work.

An asynchronous server code event and some other components can be used to implement a possible solution in this scenario. The idea is that whenever an asynchronous IPC event is started, it writes information about itself to a SmartObject. When the sub process ends, it removes the entry. If the main process needs to end, it scans the SmartObject for all the currently running sub processes that belongs to it and tells them to finish. The asynchronous server event is used in the sub-process to wait for the signal from the main process.

PROCESS PROGRAMMING AND DEBUGGING

There are many personal preferences to how developers write code, and many development teams follow company or divisional standards that make their projects run smoother. This section is meant to provide some general guidance and K2-specific methods for programming and debugging, and may or may not represent best practices within your organization.

HANDLING CONNECTIONS

Here are some pointers when using the Connection object. This object is part of the base API from which each API that makes a connection to the server inherits.

There are three scenarios in which you can connect to the server from the API.

Scenario 1: Opening and closing multiple connections

This is the most expensive way of using connections and causes the most overhead. This is because after each Open method on the connection object, the call is authenticated on the server. This scenario is typically used in a stateless environment where user context does not exist between method calls, for example in Web applications that do not store session state.

```
public static void MultipleConnections()
{
    SmartObjectManagementServer smoManagementServer = new
SmartObjectManagementServer();

    SmartObjectClientServer smoClientServer = new SmartObjectClientServer();

    string connectionString =
"Integrated=True;IsPrimaryLogin=True;Authenticate=True;EncryptedPassword=False;Host
=blackpearl;Port=5555";

    SCCConnectionStringBuilder connectionBuilder = new
SCCConnectionStringBuilder(connectionString);

    try
    {
        // Get SmartObject List
        smoManagementServer.CreateConnection(connectionBuilder.ToString());
        smoManagementServer.Connection.Open(connectionBuilder.ToString());
        SmartObjectExplorersmoExplorer = smoManagementServer.
            GetSmartObjects(SmartObjectInfoType.System);
    }
}
```

```
catch
{
    Console.WriteLine(ex.Message);
}
finally
{
    smoManagementServer.Connection.Close();
}

try
{
    // Get SmartObject
    if (smoClientServer.Connection == null)
        smoClientServer.CreateConnection();
    smoClientServer.Connection.Open(connectionBuilder.ToString());
    SmartObject smo = smoClientServer.GetSmartObject
        (smoExplorerer.SmartObjects["UMUser"].Guid);
}
catch (Exception ex)
{
    Console.WriteLine(ex.Message);
}
finally
{
    smoClientServer.Connection.Close();
}

try
{
```

```
// Get SmartObject Data
if (smoClientServer.Connection == null)
    smoClientServer.CreateConnection();
smo.MethodToExecute = "Get_Users";
smo.ListMethods[smo.MethodToExecute].Parameters["Label_Name"].Value = "K2";
smoClientServer.Connection.Open(connectionBuilder.ToString());
SmartObjectListuserList = smoClientServer.ExecuteList(smo);
}
catch (Exception ex)
{
    Console.WriteLine(ex.Message);
}
finally
{
    smoClientServer.Connection.Close();
}
}
```

Scenario2: One connection performing many operations.

This is a much more efficient way of using connections. You can share the connection object between APIs thereby making it easier to manage the connection overhead. This method should not be used in environments where network connections periodically fail or are reset.

```
public static void OneConnection()
{
    SmartObjectManagementServer smoManagementServer = new
SmartObjectManagementServer();

    SmartObjectClientServer smoClientServer = new SmartObjectClientServer();

    string connectionString =
"Integrated=True;IsPrimaryLogin=True;Authenticate=True;EncryptedPassword=False;Host
=blackpearl;Port=5555";

    SCCConnectionStringBuilder connectionBuilder = new
SCCConnectionStringBuilder(connectionString);
```

```
smoManagementServer.CreateConnection();

smoClientServer.Connection = smoManagementServer.Connection;

smoManagementServer.Connection.Open(connectionBuilder.ToString());

SmartObjectExplorer smoExplorer =
smoManagementServer.GetSmartObjects(SmartObjectInfoType.System);

SmartObject smo =
smoClientServer.GetSmartObject(smoExplorer.SmartObjects["UMUser"].Guid);

smo.MethodToExecute = "Get_Users";

smo.ListMethods[smo.MethodToExecute].Parameters["Label_Name"].Value = "K2";

SmartObjectList userList = smoClientServer.ExecuteList(smo);

smoManagementServer.Connection.Close();

}
```

Scenario3: Using sessions.

This is the most efficient way of handling connections, providing you have session state.

If you're calling client has state and can manage sessions, this is the way to use it with the connection object. The session connection timeout on the server can be set but keep in mind that this setting applies to the entire server, which means all servers hosted on that server. The default time is 20 minutes. If a session times out and you make an API call without Authenticating (Authenticate=true), you will get an exception.

```
public static void UsingSessions()

{

    SmartObjectManagementServer smoManagementServer = new
SmartObjectManagementServer();

    SmartObjectClientServer smoClientServer = new SmartObjectClientServer();

    string connectionString =
"Integrated=True;IsPrimaryLogin=True;Authenticate=True;EncryptedPassword=False;Host
=blackpearl;Port=5555";

    SCCConnectionStringBuilder connectionBuilder = new
SCCConnectionStringBuilder(connectionString);

    smoManagementServer.CreateConnection();

    smoClientServer.Connection = smoManagementServer.Connection;

    smoManagementServer.Connection.Open(_connBuilder.ToString());

    String sessionID = smoManagementServer.Connection.GetResumableSessionCookie();

}
```

```
SmartObjectExplorer smoExplorer =
smoManagementServer.GetSmartObjects (SmartObjectInfoType.System);

smoManagementServer.Connection.Close();

// Time elapses...

SmartObject smo = null;
try
{
    connectionBuilder.Authenticate = false;
    smoClientServer.Connection.Open (connectionBuilder.ToString());
    smoClientServer.Connection.ResumeSession (sessionID);
    smo =
smoClientServer.GetSmartObject (smoExplorer.SmartObjects ["UMUser"].Guid);
    smoClientServer.Connection.Close();
}
catch (Exception ex)
{
    Console.WriteLine (ex.Message);
}

// More time elapses...

if (smo != null)
{
    SmartObjectListuserList;
    try
    {
        smo.MethodToExecute = "Get_Users";
        smo.ListMethods [smo.MethodToExecute].Parameters ["Label_Name"].Value =
"K2";
```

```
smoClientServer.Connection.Open(connectionBuilder.ToString());
userList = smoClientServer.ExecuteList(smo);
smoClientServer.Connection.EndSession(sessionID);
smoClientServer.Connection.Close();
}
catch (Exception ex)
{
    Console.WriteLine(ex.Message);
}
}
```

USING CONSOLE.WRITELINE AND LOGGING

Although K2 blackpearl offers standard debugging support allowing developers to attach a Visual Studio debugger to a running K2 server and step through code, one can also use the simple WriteLine approach to output debugging statements to a K2 server running in console mode.

```
Console.WriteLine("your message here");
```

As a best practice, however, developers are encouraged to use the built-in logging framework to output messages to the selected logging configuration. This is a more robust approach and allows messages to appear in a number of different targets, such as the Windows Event Log, the K2 log file, the console and even a message queue, depending on the configuration of the logging framework.

```
string loggedInfoSource = string.Format("{0}/{1}",
    K2.ProcessInstance.Process.Name, K2.Event.Name);
K2.ProcessInstance.Logger.LogInfoMessage(loggedInfoSource , "Your message here");
```

Illustrated here is a fraction of the functionality provided by the logging framework. For more information see [KB000298 - K2 Auditing and Logging \(http://help.k2.com/kb000298.aspx\)](http://help.k2.com/kb000298.aspx).

EXCEPTION HANDLING

Exception handling is the occurrence of a condition that changes the normal flow of execution. Developers know what an exception is in code, but you should also be aware of business flow exceptions and plan for them differently. In a business flow exception, the process designer should take into account things that may change

the normal flow of the business process. Handling these exceptions within the process design allows a process administrator to receive assigned a task and take action in order to correct the business flow exception. Code exceptions are different, and what follows in this section is related to these types of exceptions.

Making use of a good logging class, whether using the one that ships with K2 blackpearl or a different one, which can be configured to log errors, warnings or traces is good practice and ensures that all exceptions are handled and displayed in the same manner. If you configure the class to log errors, it should only log errors, but if you configure to log traces, it should record all information such as errors, warnings and traces. Your log class should be written in such a way that you can easily change it to log to the Windows Event Log, a SQL Server, a log file, or email to administrator without any change in any other part of the application. Use the log class extensively throughout the code to record errors, warning and even trace messages that can help you troubleshoot a problem.

K2, by default and without explicit exception handling, bubbles all process exceptions up to a process exception handler and puts the process in an error state. The error messages in this case are written to the log targets based on the logging configuration. Depending on each process, further exception handling can be defined on the process, activity or event level. In most cases it will not be necessary to include exception handling on the activity or event levels unless specific actions are required and you do not want exceptions at these levels to bubble up to the process level.

Server events should contain proper error handling in the code to try and catch expected exception conditions (for instance during Web service calls) and to log these exceptions properly.

The following example is bad exception handling in .NET code and should be avoided:

```
catch (Exception e)
{
    myLogExceptionMethod(e);
    throw new exception("some exception message");
}
```

The following is an example that is better exception handling:


```
catch (Exception e)
{
    myLogExceptionMethod(e);
    throw new Exception("Error in Module X:", e);
}
```

This better approach ensures that the full stack trace is maintained, whereas the first code block breaks the stack trace. The "some exception message" would appear at the top of the stack trace in the first case.

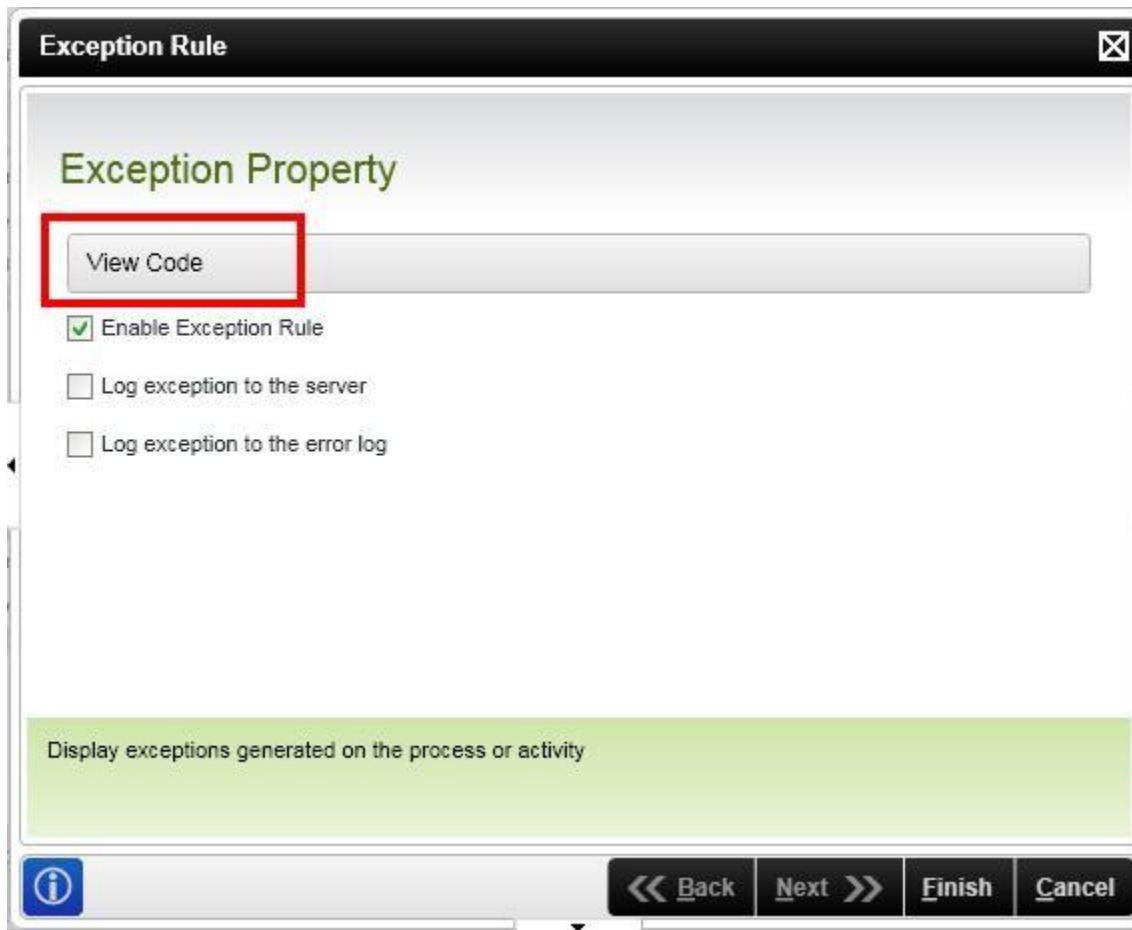
If a monitoring solution such as Microsoft Operations Manager (MOM) is in use, the error handling can be incorporated into the monitoring solution. Alternatively error notifications can be sent to K2 system administrators for action.

The most important point is to use a consistent manner for error reporting in all processes.

K2 blackpearl provides exception handling blocks at the multiple levels, similar to the different escalation levels -- Event, Activity and Process.

You can access the exception block by clicking on the icon at the top of the K2 Designer for Visual Studio canvas ()

This will open up the exception dialog. You can get to the code by clicking on 'View Code' button.



After the View Code button is clicked the XOML is displayed. Right click on the XOML canvas and select "View Code". In the code expand the Properties_ExecuteCode method.

The K2 context object exposed here provides details around the error and native workflow entity object access that is limited to ProcessInstance. In many situations it can be useful to get object level access to entities like ActivityInstance and ServerEvent, and this can be accomplished by casting the K2.ContextObject to the appropriate class. The appropriate class can be determined by checking the K2.ContextType properties.

Below is an example of getting access to the ServerEvent and ClientEvent context objects for additional exception processing. In this case the exception details are added to data fields for easier tracking and reporting:

```
private void Properties_ExecuteCode(object sender, EventArgs e)
{
    K2.AddToErrorLog = K2.Configuration.IsErrorLog;

    K2.AddToServerLog = K2.Configuration.IsServerLog;

    string ErrorType = K2.ContextType.ToString();
    string EventName = "";
    string ActivityName = "";

    Exception ex = (Exception)K2.ExceptionObject;

    string ErrorMessage = ex.InnerException.Message;

    bool Handled = false;

    switch (K2.ContextType)
    {
        case (SourceCode.KO.ContextType.ServerEvent):
            SourceCode.KO.ServerEventContext MyServerEvent =
                (SourceCode.KO.ServerEventContext)K2.ContextObject;
            ActivityName =
                MyServerEvent.ActivityInstanceDestination.Activity.Name;
            EventName = MyServerEvent.Event.Name;
```

```
try
{
    MyServerEvent.ActivityInstanceDestination.ActivityInstance.
        DataFields["ErrorMessage"].Value = ErrorMessage;
    MyServerEvent.ProcessInstance.DataFields["Last Error Message"].
        Value = "Activity: " + ActivityName + " Error: " + ErrorMessage;
    MyServerEvent.ExpireActivity(ActivityName);
    Handled = true;
}
catch (Exception exception)
{
    Console.WriteLine("Exception Rule Error: " + exception.Message);
}
break;

case (SourceCode.KO.ContextType.ClientEvent):
    SourceCode.KO.ClientEventContext MyClientEvent =
        (SourceCode.KO.ClientEventContext)K2.ContextObject;
    ActivityName =
        MyClientEvent.ActivityInstanceDestination.Activity.Name;
    EventName = MyClientEvent.Event.Name;
    try
    {
        MyClientEvent.ActivityInstanceDestination.ActivityInstance.
            DataFields["ErrorMessage"].Value = ex.Message;
        Handled = true;
    }
    catch (Exception exception)
    {
```

```
        Console.WriteLine("Exception Rule Error: " + exception.Message);
    }

    break;
}

// if this wasn't handled explicitly then handle the normal K2 way
if (!Handled)
{
    K2.AddToErrorLog = true;
    K2.AddToServerLog = true;
}
}
```

PROJECT DEPLOYMENT

As mentioned in the K2 blackpearl Installation, Configuration and Security Best Practices section, you should use MSBuild tasks to deploy to QA/Test and Production environments. This is not only a best practice, it also ensures that updated references are copied or added to the global assembly cache (GAC) upon successful builds, and it does not require developers to have access to these environments.

The deploy package can be created and sent to the server team who can then deploy the package to the relevant environment. The K2 deployment package will only deploy processes and SmartObjects, not, for example, user interfaces, reports, service objects, custom environment library fields, roles, working hour zones, and permissions. The developer will need to create a separate deployment method for these components. Refer to the following K2 KB article on creating deployment packages: <http://help.k2.com/kb000188.aspx>

Note: At design time, SmartObjects are associated to Service Objects using the service instance GUID. If the Service Instance GUID differs between environments, deployment errors will occur. It is therefore recommended to set the Service Instance GUID to the same value for every environment when using the Broker Management tool to configure service instances. There are also tools available, such as the **Service Object GUID Updater** (<http://www.k2underground.com/k2/ProjectHome.aspx?ProjectID=25>), on the K2 blackmarket on [k2underground.com](http://www.k2underground.com) to synchronize Service Instance GUIDs as well as a Knowledge Base article -- KB000250 - SmartObject GUID Synchronization (<http://help.k2.com/articles/kb000250.aspx>). The Amazing SmartObject tool is also made available with K2 blackpearl 0807 and later releases, and should be used to test SmartObjects. It is called the SmartObject Service Tester and is located in the ServiceBroker folder.

CONCLUSION

There are many practices outlined in this whitepaper that will assist you in identifying, designing, deploying and maintaining your processes. This whitepaper may be updated in the future with more best practices. If you have a best practice you would like to share, please post to the K2 blackpearl forums on K2underground.com and mention that you would like it to be included in this whitepaper.

GLOSSARY

Term	Definition
Action	A specific choice or response made by a user in the course of a client event.
Activity	Step in a K2 blackpearl Process, containing one or more Events
AD	Active Directory
API	Application Programming Interface
Association	A relationship between two SmartObjects. This is achieved by mapping fields together. For example with an object 'Employee' and an object 'Leave Application', we could relate the two objects by mapping the 'Employee' object's 'Employee Number' field with a field of the same name on the 'Leave Application' object.
BDC	See <i>Business Data Catalog</i>
BPA	Business Process Automation
BPM	Business Process Management
Business Data Catalog	A catalog of business applications that are of interest to a SharePoint 2007 User. The K2 BDC for SharePoint 2007 enables the data contained within a K2 SmartObject to be available via a SharePoint Site Web Part.
Business Process	Sequence of tasks done in a predefined order in the real world.
Business Rule	A rule, applicable to a workflow, stating behavior in a given set of circumstances
Data Field	Process data fields and activity data fields facilitate the collection, communication and management of information required in the process
Data Field – Hidden	Prevents the Data Field from being shown in the Object Browser.
Data Field – On	If selected, this ensures that K2 Server only loads Data Fields when and if

Demand	required either by reading a value or setting a value in a data field.
Data Field – Keep Audit	This option enables you to keep track of the changes made during the execution of the workflow process and is selected by default.
Data Field – Keep Log	Keeps a log of the data field
DoD	Data on Demand – see <i>Data Field – On Demand</i>
EAI	Enterprise Application Integration
Event	Smallest action in a K2 blackpearl Process, executed serially within an Activity
Event Bus	System events notification service
IPC	Inter-Process Communication
K2 blackpearl Server	Server side components for managing and operational requirements of the K2 blackpearl environment.
K2 Designer for SharePoint	Microsoft SharePoint design environment for K2 processes
K2 Designer for Visio	K2 blackpearl integration with Microsoft Visio 2007.
K2 Designer for Visual Studio	The K2 blackpearl design environment which is fully immersed in the Visual Studio development environment.
K2 Configuration Manager	Environment configuration application.
K2 Management Console	Security and permissions are maintained in the K2 Management Console which surfaces in K2 Workspace.
K2 MOSS Components	Components enabling SharePoint integration.
K2 WorkList	List of work items assigned to some specific user or group.
K2 Workspace	Browser based interface that enables users to manage their work list, view and build reports that can be used to manage business processes.
K2 WSS Components	Components enabling WSS integration.
Line	In K2 blackpearl, lines contain rules (business rules) and link Activities together
Line Rule	An optional piece of logic that determines the route followed within the process as it executes.

Method	An action provided by a SmartObject. For example 'Load' or 'Calculate Tax'. This links back to a method provided by a ServiceObject.
MOSS	Microsoft Office SharePoint Server
MSDTC	Microsoft Data Transaction Coordinator
Outcome	The Outcome requires a rule or logic that allows the rule to resolve to true (succeed). The rule defines the conditions required for the rule to succeed.
PM	Project Manager
Process	A single design of a Workflow in K2 blackpearl (interchangeable with the word "Workflow")
Property	An item of data provided by a SmartObject. For example 'Name' or 'Vehicle Colour'. This may link back to a property provided by a ServiceObject, or it may be a piece of data stored in SmartBox
ServiceObject	An object provided by the K2 blackpearl SmartBroker that provides one or more back-end services. These could include access to data systems, standard calculation or processing systems or code, or web services
Service-oriented Architecture	A service-oriented architecture is a collection of services that communicate with each other. The services are self-contained and do not depend on the context or state of the other service. They work within distributed systems architecture.
SmartBox	An Out-Of-The-Box service supplied with K2 blackpearl server, used for storing SmartObject data that has no backing store.
SmartBroker	A process hosted by the K2 blackpearl HostServer that provides ServiceObject and SmartObject capabilities and organisation.
SmO	SmartObject (see <i>SmartObject</i>)
SmartObject	An object provided by the K2 blackpearl SmartBroker that provides a business centric view of workflow or back-end data. For example a SmartObject called 'Employee' might pull data from AD, the company payroll system and a database of employees, to provide an overall picture of each employee.
SO	ServiceObject (see <i>ServiceObject</i>)
SOA	See <i>Service-oriented Architecture</i>
SPS	SharePoint Portal Server
Task List	See <i>K2 WorkList</i>



Topology	The arrangement of networked elements and the interconnections between them.
UM	User Manager
ViewFlow	A diagram within the K2 Workspace (but could be embeddable elsewhere) that graphically shows the status of a given process instance.
Workflow	A single design of a process with sequential steps in K2 (interchangeable with the word "Process")
WCF	Windows Communication Foundation
WF	Windows Workflow Foundation
WSS	Windows SharePoint Services